
Introduction To The Sas Macro Language Lex Jansen

*Introduction
To The Sas
Macro
Language
Lex Jansen*

*Downloaded
from
evoluir.abar.org.br
by Dakota &
Greyson*

UNLOCKING THE POWER OF SAS: AN INTRODUCTION TO THE SAS MACRO LANGUAGE WITH LEX JANSEN

For anyone who has spent time writing SAS programs, the transition from repetitive, hard-coded scripts to dynamic, reusable code is a career-defining moment. At the heart of this transformation lies the SAS Macro

Language—a powerful tool that extends the capabilities of Base SAS and allows programmers to write code that writes code. While official documentation provides the syntax, the real-world wisdom often comes from experienced practitioners. Few names are as synonymous with practical macro education as **Lex Jansen**. This article provides a comprehensive **introduction to the SAS macro language**

Lex Jansen style, blending foundational concepts with the practical insights that Jansen's work has popularized across the global SAS community.

Whether you are a data analyst, a statistician, or a seasoned programmer new to SAS, understanding macro language will elevate your efficiency, reduce errors, and open doors to advanced automation. Let us begin this journey by exploring what macros are, why they matter, and how Lex Jansen's contributions have made this topic accessible to thousands.

WHAT IS THE SAS MACRO LANGUAGE?

The SAS Macro

Language is a text-processing subsystem within SAS. It is not a traditional programming language like Python or R; instead, it generates SAS code before that code is compiled and executed. This means you can write a small piece of macro code that expands into a much larger block of SAS statements, tailored dynamically to your data and parameters.

Key capabilities include:

- **Dynamic code generation:** Create program steps that adapt based on dataset names, variable lists, or runtime conditions.
- **Parameterization:** Reuse the same logic for

ENDURING Different inputs
without editing
the program.

- **Automatic
avoidance of
repetition:**

Write once, run
many
times—perfect
for monthly
reports, batch
processing, or
iterative
analyses.

Macro programming is
often considered the
bridge between basic
SAS and professional-
grade applications.
Without it, any change
in a variable name or
file path forces you to
edit every line. With
macros, you simply
change a single macro
variable, and the entire
program updates.

WHY LEX JANSEN? A GUIDE TO HIS

INFLUENCE

If you search for SAS
macro tutorials online,
you will inevitably
encounter the name
Lex Jansen. A long-time
SAS user, presenter at
conferences such as
**SUGI (SAS Users
Group International)**
and **PharmaSUG**,
Jansen has compiled an
extensive library of
papers, code samples,
and presentations. His
website,
lexjansen.com, is a
treasure trove for
anyone seeking
practical macro
examples, from basic
%LET statements to
complex nested
macros.

What sets Lex Jansen
apart is his ability to
explain abstract macro
concepts with
relatable, real-world
examples. He does not

just show you the syntax—he shows you *why* you would use it. His papers on macro quoting, conditional logic, and iterative loops remain go-to references for SAS programmers at all levels. This article draws heavily on the principles Lex Jansen has championed: clarity, modularity, and the relentless pursuit of automation.

Key Contributi ons of Lex Jansen

1. **Comprehensive macro catalogs:**

Curated examples ranging from simple variable substitution to advanced metadata-driven programs.

2. **Error-proofing techniques:**

Detailed guides on handling special characters, quoting, and debugging macros.

3. **Community building:** Freely sharing code and presentations, ensuring that the macro knowledge is not locked behind paywalls.

For anyone seeking an **introduction to the SAS macro language Lex Jansen** approach, the first step is to embrace his mindset:

macros are not just shortcuts; they are a design philosophy for robust, maintainable SAS applications.

CORE BUILDING BLOCKS OF THE MACRO LANGUAGE

Before diving into syntax, it is essential to understand the two fundamental components: **macro variables** and **macro programs**.

Macro Variables

Think of macro variables as placeholders for text. Unlike data step variables, they exist outside the DATA step and are resolved during the macro processing stage. They can hold any text, from

a simple number like 2025 to a long block of SAS statements.

- **Global vs. Local:** Global macro variables are available everywhere in your session; local ones exist only within a specific macro.
- **Automatic macro variables:** SAS provides built-in variables such as &SYSDATE (current date) and &SYSLAST (last created dataset).

Example:

```
%LET year = 2024;  
%PUT The report  
year is &year.;  
This resolves to: "The  
report year is 2024."
```

Macro Programs

A macro program is defined using %MACRO and %MEND statements. It encapsulates a set of SAS statements and optionally accepts parameters. When you invoke a macro, SAS replaces the macro call with the generated code, then executes it.

Basic structure:

```
%MACRO
mymacro(param1,
param2);
  data &param1.;
    set &param2.;
    * ... more
statements ...;
run;
%MEND mymacro;
```

Invocation:

```
%mymacro(work.out
, sashelp.class);
```

This pattern—parameterization and reuse—is

exactly what Lex Jansen emphasizes in his tutorials. He often demonstrates how a single macro can handle dozens of similar tasks, drastically reducing code volume and potential errors.

GETTING STARTED: YOUR FIRST MACRO EXAMPLE

Let us walk through a realistic scenario: you need to generate summary statistics for multiple datasets every month. Without macros, you would copy and paste the same PROC MEANS code and change the dataset name manually. With macros, you create a reusable template.

Step 1: Define the Macro

We want a macro that takes a dataset name and a variable list. It will produce a summary table.

```
%MACRO summarize(dsn,
vars);
  proc means
data=&dsn. n mean std
min max;
  var &vars.;
  title "Summary
for &dsn.";
run;
%MEND summarize;
```

Step 2: Invoke It

Now you can call %summarize(sashe
l.p.class, age
height weight) and
later

%summarize(sashe
l.p.heart,
cholesterol
weight). Each call
generates a
customized PROC
MEANS step. This is a
trivial example, but the
principle scales to
complex report
generation, data
validation, and even
entire ETL pipelines.

Lex Jansen's
**introduction to the
SAS macro language**
often starts with such
simple examples to
build confidence, then
gradually introduces
more advanced topics
like %DO loops, %IF-
%THEN logic, and
quoting.

MASTERING CONDITIONAL LOGIC AND LOOPS

Macro language
supports its own set of
programming

constructs that operate during code generation. These are different from DATA step conditionals; they control what code gets produced, not how data is processed.

%IF-

%THEN /

%ELSE

Use these to conditionally generate SAS statements. For example:

```
%MACRO reporter(dsn);
  %if %upcase(&dsn.)
    = SASHELP.CLASS %then
  %do;
    proc print
      data=&dsn.; run;
  %end;
  %else %do;
    proc means
      data=&dsn.; run;
  %end;
%MEND reporter;
```

Here, depending on the

dataset name passed, the macro produces either a PRINT or a MEANS step. Lex Jansen often highlights the importance of using %UPCASE or similar functions to make comparisons case-insensitive, a subtlety that can save hours of debugging.

%DO

Loops

Iterative macro loops allow you to repeat code generation. For instance, to run the same analysis on multiple years stored in separate datasets:

```
%MACRO loop_years;
  %do i = 2020 %to
    2024;
    %summarize(data_&i.,
      sales profit);
  %end;
%MEND loop_years;
```

This generates five

separate PROC MEANS calls. Such loops are a hallmark of macro automation and are heavily featured in Lex Jansen's papers on batch processing.

THE QUOTING CHALLENGE: WHAT LEX JANSEN TEACHES US

One of the trickiest aspects of macro programming is **quoting**. Special characters like &, %, or semicolons can be misinterpreted by the macro processor. Lex Jansen has written extensively on this topic, offering practical solutions using %STR, %NRSTR, %SUPERQ, and other functions.

A typical pitfall: passing a macro variable that contains a period or ampersand. Without proper

quoting, the macro processor tries to resolve text that should remain literal. Jansen's guidance: "When in doubt, quote it out." His examples often show how to safely pass filenames, titles, or SQL clauses without unintended resolution.

Key functions to remember:

- %STR - masks tokens until execution.
- %NRSTR - masks everything until execution, including & and %.
- %BQUOTE - masks at compile time for tricky characters.

Using these correctly can make the difference between a stable macro library and one that breaks

unpredictably.

PRACTICAL TIPS FROM LEX JANSEN'S PLAYBOOK

Drawing from his decades of experience, here are actionable tips to improve your macro coding:

1. **Start simple and build incrementally.** Write a macro that works for one specific case, then add parameters and conditions. Do not try to build a universal monster on the first attempt.
2. **Use explicit scope for macro variables.** Avoid relying on global variables created inadvertently.

Declare %GLOBAL or %LOCAL explicitly to prevent collisions.

3. **Comment liberally.** Macros can become cryptic. Use %* for macro comments that are not compiled.
4. **Test with %PUT statements.** Insert %PUT _LOCAL_; or %PUT _GLOBAL_; during development to see what variables are in scope and what values they hold.
5. **Keep macros short.** A macro that does one thing well is easier to debug and reuse. Lex Jansen's own published

macros rarely exceed 100 lines; they are modular and focused.

EXPLORING LEX JANSEN'S ONLINE RESOURCES

To deepen your **introduction to the SAS macro language**, visiting lexjansen.com is

essential. The site organizes papers by topic: macro design, dynamic programming, reporting, and data management. Notable papers include:

- *“Writing Macro Code That Writes Code”* – a guide to intermediate and advanced techniques.